The Potential of Augmented Reality for Computer Science Education

Dmitry Resnyansky  
School of Information Technology and Mathematical Sciences  
University of South Australia  
Mawson Lakes, Australia  
dmitry.resnyansky@mymail.unisa.edu.au

Emin Ibili  
Computer Education and Instructional Technology  
Aksaray University  
Aksaray, Turkey  
eminibili@aksaray.edu.tr

Mark Billingham  
School of Information Technology and Mathematical Sciences  
University of South Australia  
Mawson Lakes, Australia  
mark.billinghurst@unisa.edu.au

Abstract—Innovative approaches in the teaching of computer science are required to address the needs of diverse target audiences, including groups with minimal mathematical background and insufficient abstract thinking ability. In order to tackle this problem, new pedagogical approaches that make use of technologies such as Virtual and Augmented Reality, Tangible User Interfaces, and 3D graphics are needed. This paper draws upon relevant pedagogical and technological literature to determine how Augmented Reality can be more fully applied to computer science education.

Keywords—program visualisation, augmented reality, computer science education, tangible interface, medium, multimodal

I. INTRODUCTION

This paper reviews how Augmented Reality could be used for Computer Science education. Computer Science (CS) education requires addressing several challenges. First, students need to grasp abstract theory, including concepts of loops, conditionals, array-based lists, linked lists, hash tables, and data structures such as stacks and queues, etc. [1][2]. It is also necessary to understand the development environment used for programming operations [3][4]. Finally, students must learn the ability to apply algorithms to solve real-world problems, to think like programmers, and to work like programmers in the field – for instance, to be able to collaboratively work on a section of code [5].

There is a common consensus that teaching tools and learning environments involving new technologies are needed throughout the entire continuum spanning early childhood to higher education [6]–[8]. Considerable potential exists for technologies such as Virtual Reality (VR), Augmented Reality (AR), Tangible User Interfaces (TUI), and projection displays, within the educational sphere.

Augmented Reality (AR) is a method of combining the virtual and physical environments to form a “middle ground” between a virtual interface and the tangibility of the real world [9]. It enables innovative pedagogical approaches (active learning, action-oriented teaching and learning, collaborative learning, and flipped classroom learning [10][11]) in a variety of areas – from literacy to STEM to the arts, as well as supporting interactive learning within different settings (eLearning, mobile learning, blended learning, and so forth). Nevertheless, the potential of AR for teaching abstract formal subjects, including CS is yet to be revealed and utilised [12].

This paper draws upon relevant educational and technological literature in order to understand how AR can be more fully applied within the field of computer science education. AR bridges the digital and the real, and so may contribute to more effective outcomes in computer science education by enabling a more transparent representation of computing concepts, processes, and programmer “real-world” activities.

II. AUGMENTED REALITY AS AN INTERFACE MEDIUM: BRIDGING THE DIGITAL AND THE REAL

At its core, AR technology relies on overlaying a variety of virtual information seamlessly onto the real world [9] in a multitude of forms ranging from text-based and multi-modal interfaces to two- and three-dimensional hybrid interfaces. The defining features of an AR interface are: (1) simultaneous display of real and virtual images to the user, (2) real-time interaction between the user and the graphical interface, and (3) a mechanism for tracking and registration of the user’s position and viewpoint within a 3D virtual space [13].

AR technology continues to build upon successful applications in specialised areas that require the incorporation of digital information within real world settings. Notable examples include: systems that enable doctors to view medical data in real time over a patient’s body [14]; systems that guide tourists through archaeological environments by displaying a location’s artifacts and history [15]; systems that simulate design, construction, and engineering solutions for architectural projects [16]–[18]; and so on – all providing the user with real time image rendering of objects, concepts, and cues directly onsite.

The features that make AR attractive in industry and entertainment similarly prove beneficial in education. AR tools focusing on teaching natural or “hard” sciences such as physics, biology, chemistry, and engineering have demonstrated benefits for student learning [13][19]. For example, in the Augmented Chemistry environment [20] a student is able to engage with simulations of physical objects (molecules; chemical elements), activities (picking and mixing chemical substances), and processes (chemical reactions). As a
tool for teaching natural science, Augmented Chemistry builds upon a long tradition of using physical models as teaching aids [21].

AR may also be applied in the teaching of early childhood literacy and reading [22][23]. ARBlocks is one such example, making use of AR, TUIs and Gesture-based mediums in order to equip teachers and young students with a dynamic block platform for educational activities through the modality of 2D visualisation, typography and projection onto physical objects.

The educational application of AR is not reserved exclusively for edutainment and disciplines that study empirical reality, and is gradually being recognised as a tool to enhance learning of formal sciences [12][24][25]. For example, the geometry education system demonstrated in [26] uses AR representations of shapes and configurations that mirror real life physical objects without the disadvantages of a lack of physical materials. At the same time, it retains all of the advantages of additional visual cues and information overlays to enhance learning for students. Visualisation methods like this are a valuable asset in aiding visual reasoning for learners of geometry in such cases as area formula derivations in two- and three-dimensional shapes; they can also facilitate the skill of “seeing to think” by shaping ideas in the learner’s mind [25].

III. AUGMENTED REALITY AND TUIs IN COMPUTER SCIENCE EDUCATION

Attempts to introduce AR and TUI technology into the realm of computer science education have resulted in the development of tools that primarily support the teaching of core programming concepts to young learners (see Table 1). Through tools like AR Scratch [32], AlgoBlocks [27], ToonTalk [33], and Quetzal [7], children are taught aspects of control flow structure and logical operation with variables, among other concepts. These tools present programming concepts through visual means (colour-coding), as well as through physical (slot-based 3D wooden shapes; plastic triangles) and spatial-kinetic analogs of abstract concepts and instructions (branches, loops, etc.). In adopting AR and TUIs, such tools draw upon the constructivist approach in developmental psychology and a pedagogical tradition of using manipulatives (building blocks, toys, etc.) in early childhood education [8], employing the building block metaphor to emphasise the nature of coding as a collaborative activity.

Within research on educational technology, the potential of harnessing existing pedagogical practice through AR- and TUI-based tools is primarily seen as reducing cognitive load while encouraging student engagement through activities that result in heightened feelings of fun and excitement [23][34][35]. However the contribution of these tools to the absorbing of difficult-to-grasp concepts, like those encountered in typical programming learning environments, is still open for discussion. Some researchers doubt the capacity of such tools to facilitate students’ transition to text-based languages [36], and to foster the ability to solve real-world problems and think as a programmer [23].

For example De Raffaele et al. [12] write that “Whilst Tangible User Interface (TUI) systems have successfully been developed to edutain children in various research, TUI architectures have seen limited deployment in more complex and abstract domains” [12, p. 535], such as artificial intelligence and programming.

IV. TOOLS FOR TEACHING AND LEARNING COMPUTER SCIENCE: THE MEDIUM/MODALITY PERSPECTIVE

Programming teaching tools can be approached in different ways. In educational literature they are commonly approached in terms of the target user group, curriculum, type of the computer language taught and in terms of their ability to engage students in independent and/or collaborative activities, as well as in their usability [37]. Developers tend to highlight technological features of the interface, preferring to focus on such aspects of educational tools as advantages and disadvantages of display systems, tracking devices, and so on [38]. Those perspectives can be brought together within a framework that interprets the interface medium as a channel of learners’ interaction with the content to be learned, and the representation modality as a means of the construction of the content.

Table 2 shows that many well-known programming teaching tools use GUI-based layouts. Among them are applications like Jeliot, JHAVÉ, Alice, BlueJ, JFLAP, Scratch, and TRAKLA2, each employing a unique set of visualisation methods in order to facilitate learning of core computer science concepts. Many share a common goal of teaching the student an OOP language, such as Java, by visually representing algorithm and program structures, and concepts involving objects, classes and methods, parameters, and so on.

As a rule, the means of visual representation within many Algorithm Visualisation (AV) and Program Visualisation (PV) environments correspond to the nature of the elements being represented. In accordance with traditions in program visualisation [51], abstract computer science concepts are communicated through abstract, highly symbolic modalities such as graphs, trees, network diagrams, flowcharts, branches, colour-coding, etc.

Symbolic modalities, while more effective than pure text in communicating formal relationships and complex technical configurations, are still considered insufficient by educators due to their abstract and static nature. A familiar problem for IDEs that employ abstract visual symbolism is one of intelligibility. For example, in cases representing iterations of program code through a two-dimensional modality such as a network diagram, comprehension becomes a challenging task even for experienced programmers [3].

Within 2D modes of visualisation, dynamic processes can be represented only as static structures of relationships between co-existing elements [52]. Visualised animation is commonly seen as a solution enabling designers of educational environments to represent programming as a dynamic process, and to capture the evolution of the program [53]. In some cases, cartoon-like depictions that represent programming concepts are used in an effort to make comprehension more intuitive. For instance, a memory stack may be represented by an image of
TABLE I. TUI/AR-BASED TEACHING PROGRAMMING TOOLS

<table>
<thead>
<tr>
<th>Tools</th>
<th>Programming Concepts</th>
<th>Representation Modalities</th>
</tr>
</thead>
<tbody>
<tr>
<td>AlgoBlocks</td>
<td>Flow-of-control structures and operations</td>
<td>Physical (slot-based blocks)</td>
</tr>
<tr>
<td></td>
<td>Operations with variables (branches, loops, etc.)</td>
<td>Visual (colour-coding)</td>
</tr>
<tr>
<td></td>
<td>Coding as collaborative activity</td>
<td>Spatial-kinetic</td>
</tr>
<tr>
<td>Osmo</td>
<td>Basic programming concepts (actions, sequences, subroutines, nested repeats, synchronisation, loops)</td>
<td>Physical (slot-based plastic shapes)</td>
</tr>
<tr>
<td></td>
<td>Program debugging</td>
<td>Visual (color-coding)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Spatial-kinetic</td>
</tr>
<tr>
<td>Tica</td>
<td>Basic programming concepts of control flow and statements (loops, conditionals, etc.)</td>
<td>Physical (slot-based plastic shapes)</td>
</tr>
<tr>
<td></td>
<td>Algorithmic concepts</td>
<td>Visual (iconic images; colour-coding)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Spatial-kinetic</td>
</tr>
</tbody>
</table>

TUI with AR compiler

| Code Bits   | Basic control flow concepts (commands, functions) | Physical (slot-based paper shapes) |
|            |                                                    | Visual (symbolic images, colour-coding) |
|            |                                                    | Spatial-kinetic                |

GUI with AR compiler

| AR Spot     | Basic concepts (variables, operators, etc.)       | Visual (slot-based 2D shapes; iconic images; colour-coding) |
|            | Algorithm structure (conditionals, loops, etc.)  |                                                            |

AR and related technologies have a potential to answer these challenges by changing pedagogical strategies and enriching the experience of programming. The idea of using 3D spatial attributes of AR to simplify complex coding concepts is founded in several key factors: (1) the readiness of users to respond to visual information over abstract and text-based material; (2) the added element of interaction, allowing the user to simultaneously inspect a piece of code both as separate and whole components, facilitating in greater understanding of the code’s structure [57]; and (3) the adaptability of AR interfaces in dealing with concepts through spatial relationships. Presenting a student with visually-engaging 3D nodes and connections to be interacted with could help those who feel overwhelmed by the abstract nature of code. Problems resulting from the lack of fundamental knowledge and necessary skills may also be addressed via the use of tools such as ARcadia [58] or Apple’s Swift Playgrounds that enable a more transparent representation of computing processes. ARcadia is a prominent example of an AR-supported environment that may assist learners in this regard. By fusing the existing Scratch visual coding platform with augmented graphics on marker objects, ARcadia allows users to produce interactive AR prototypes with relative ease. Likewise, Swift Playgrounds has made a recent addition to its Swift-based code visualisation software with users now able to output visually-engaging AR content through their iPad. The system even offers a free curriculum to help integrate AR-supported code learning into schools’ ICT programs. By working on such platforms, learners rely intuitively on a spatial and visual awareness of coded objects within tangible, 3D environments, allowing them to

wooden stacks; however, such representations have been less effective in communicating in-depth concepts involving sorting algorithms [4]. Vegh and Stöffová [4] suggest that such in-depth understanding can be achieved with more detailed, micro-level animations. However, this solution may also be problematic due to the limitations of 2D visualisation and GUIs as a medium of interaction with the program code. Alternatively, tools that employ current computing technology to visualise these processes and structures in a three-dimensional and dynamic way could be considered, so that the various states of allocation and movement of memory are made transparent.

Existing empirical studies of the effectiveness of AV/PV tools suggest the importance of the availability of explanations throughout the course of using such tools. For instance, Ma et al. [54] explored the effect of using Jeliot on students’ development of mental models of programming concepts, such as conditionals and loops, scope and parameter passing, and object reference assignment, finding that the tool proved more effective for simple concepts rather than complex ones such as object references assignments. Moreno et al. [55] note that visual content may have an opposite effect on novice programmers by confusing their comprehension of a visual representation with the meaning of the represented programming concepts. The need of students to understand the meaning behind visualisations of complex subjects may imply the decisive role of explicit instruction in comprehending the graphical notation [56].
form cognitive relationships between the interaction of abstract programming concepts and their concrete result.

An example of how AR could be used for teaching computer science, could include the development of an AR application that utilises 3D models for visualising a code structure. Such a model, with which the student may freely interact – combined with a two-dimensional on-screen text editor through an augmented GUI – may provide an additional layer of interactivity by updating in real-time in accordance with textual code information input by the user thus allowing them to understand the way in which changes can present on a visual model. This application of 3D visualisation should be employed in tandem with AR technology due to the presence of real-life interfaces (keyboard and mouse) necessary for the learner to simultaneously navigate the interface, the model, and the GUI text editor with minimal division of attention between the three. Employing such a system may greatly aid in enhancing learners’ and educators’ ability to respectively absorb complex information and communicate fundamental computer science concepts. Given the complex, multimodal nature of such applications of AR for education, their use should be supported by empirical data based on qualitative

<table>
<thead>
<tr>
<th>Tools</th>
<th>Programming Concepts</th>
<th>Representation Modalities</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Interactive game-based environments</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Scratch</td>
<td>Basic concepts (variables, operators, etc.); algorithm structure (conditionals, loops, etc.)</td>
<td>Visual (iconic images; colour-coding; slot-based 2D shapes)</td>
</tr>
<tr>
<td>Multimedia programming tool [39]</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Kodu Game Lab</td>
<td>Basic programming concepts (actions, conditionals, sub-routines, etc.)</td>
<td>Visual (visual metaphors (pages, tiles); cartoon iconography)</td>
</tr>
<tr>
<td>Visual programming integrated development environment (IDE) enabling children to primarily learn creative problem solving and storytelling, as well as some programming concepts by designing games [40]</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ToonTalk</td>
<td>Basic concurrent programming concepts (objects, classes, methods, constants, etc.); procedures (computation, routines/sub-routines, actor spawning/termination, etc.); operations (building, execution, debugging)</td>
<td>Visual (Concrete 2D, animated 2D, cartoon images (boxes, nests, birds, robots, construction block, magic wands, sensors, scales, trucks, bombs, houses, cities, etc.)); Audio</td>
</tr>
<tr>
<td>Web-based and stand-alone IDE for learning basic OO programming concepts and logic, and for improving problem-solving skills through puzzles [41]</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Stack Em Up</td>
<td>Data structure concepts (memory stacking)</td>
<td>2D animated visualisation of memory stacking through metaphor of Tower of Hanoi game</td>
</tr>
<tr>
<td>Game-based environment for learning basic computing concepts, as well as increasing learner motivation and cognitive capability [42]</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Alice</td>
<td>Objects, methods, and algorithms, parameters, as well as basic program structure</td>
<td>Text-based Linguistic command buttons and blocks</td>
</tr>
<tr>
<td>Visual programming tool facilitating students’ transition to Java language [43]</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

| **Visualisation and animation tools** | | |
| VILLE | Programming concepts (variable, statements, expressions, methods, etc.); control structures (selections, loops, etc.); debugging (execution tracing through code stack, etc.) | Visualisation: Diagrams (flowchart) Text: accompanied by explanatory text (code examples) |
| Language-independent program visualisation tool for learning core computer science concepts [44] | | |
| TRAKLA2 | Algorithm structure | Visualisation (graphs, trees) |
| Java-based tool to develop algorithm visualisations for both students and instructor [45] | | |
| JHAVE | Pseudocode representing algorithm solution of problem in visual step-by-step form | Visualisation (network structure, flowchart); slide show |
| Java-based tool for visualising algorithm states and creating slideshows by instructors for use by students [46] | | |
| Jeliot | Program source code and behaviour of an actual program, rather than an abstract algorithm, as sequential process | Visualisation/animation (network structure) |
| Program visualisation tool to help teach Java programming to high school students [47] | | |
| BlueJ | Program structure (class relationships) | Visualisation (static network structure; class diagram): interactive object/class creation and testing |
| Java development environment for aiding text-based OOP language learning [48] | | |
| JFLAP | Behaviours of formal computer language and automata concepts | Visualisation (static network structure): interactive graphic editor |
| Java-based tool for helping teach formal language and automata theory [49] | | |

| **3D simulation** | | |
| 3D LED Cube | OOP (JavaScript) concepts (loops, conditionals, arrays, recursion) | 3D simulation and physical model equivalent (Arduino LED micro-controller); abstract visual patterns and effects |
| 3D browser-based environment for learning programming concepts through visualisation using Arduino 3D LED board [2] | | |
| Processing | Basic OOP concepts (objects, classes, methods, etc.); dynamic algorithm structure; logical operations; statements (branches, loops, etc.) | Real-time animated visualisation with text-editor, with 2D rendering and 3D OpenGL support |
| Java-based development environment for coding visualisations using OOPL and for use as learning tool [50] | | |
studies of relevant learner groups. Such research may benefit by focusing on the cognitive and behavioural theories that serve to inform the decisions behind environment design. This may include studies into the collaborative aspects of using AR-supported learning tools in CS education, as such areas are often left unexplored, particularly when discussing the effect of AR educational tools for collaborative learning of complex code.

The positive effects of applying AR/TUI tools in the CS classroom have been observed in research, particularly for indicators such as user satisfaction, general levels of enthusiasm and learning outcomes. A survey study of university students of varying CS backgrounds learning OOP concepts by means of a mobile AR-supported e-learning system reported significant increases in cases of academic improvement and understanding of abstract technical concepts [59]. Similar research into systems designed to aid learning of control structures have proven to have positive effects for student learning [60]. In a study of 54 students of introductory CS subjects with different theoretical and practical backgrounds, 100% of participants demonstrated correct use of control structures in a post-test (compared to 70% in a pre-test). All students attributed the use of AR to a better understanding of the topic, with the overwhelming majority expressing enjoyment in a satisfaction survey of their experiences. Del Bosque et al. [61] have studied the effect of AR on failure rate dynamics amongst students in programming education. It has been found that failure rates decreased by half for subjects involving structured programming concepts, and students’ performance, achievement and motivation levels saw an increase as a result of introducing an AR application into the classroom.

V. CONCLUSION AND FUTURE RESEARCH

The development of Computer Science educational tools that are able to support information absorption and retention requires addressing many direct and indirect aspects of education – including pedagogical, psychological, cognitive and technological [6][32]. It also needs to be informed by empirical research focusing on the cognitive and social effects of different types of interfaces [21]. From the developer’s perspective, it is necessary to understand what aspects of the educational process or elements of the educational content of the tool are to be represented and supported, and how better to use the potential of different mediums and modalities for achieving specific teaching purposes.

This paper maintains that, in order to fully use the affordances of AR – in particular its ability to bring together the digital and the real – it is necessary to re-examine the relationship between the medium/modality on the one hand and the concept communicated, or the object acted upon, on the other hand. In order to develop more advanced systems, it may be useful to draw upon the knowledge gained from the application of AR in broader areas.

Tools developed with the teaching of computer science concepts in mind differ to those focusing on tangible or concrete objects such as medical data, molecules, architecture, and alphabet cubes. The latter types of objects can be adapted relatively easily and with a high degree of efficacy to modalities based in AR and TUIs. Teaching ‘tangible’ subjects (physics, astronomy, chemistry, archaeology, etc.) naturally presupposes overlaying direct information through realistic visual representation or using tangible instruments to manipulate virtual simulations, rather than on abstract metaphor or symbolism, as is the case with concepts of various algorithm structures used in programming. A 3D representation may be needed to allow the user to observe processes involving memory allocation within stacks and heaps, the use of pointers, creation and destruction of variables, function calls, and so on from multiple angles. The ability to “experience” programming processes as an interplay of the digital and material planes through the mode of AR can further enhance understanding of abstract concepts, data structures, and the evolving nature of dynamic memory.

The survey of literature conducted in this paper suggests that further research into the application of augmented reality in CS education needs to be conducted. In particular, such research needs to focus on the potential of AR/TUIs on collaborative learning in this field. Collaborative CS learning “introduces new challenges and possibilities that are different from the ones related to individual learning with visualisation” [62, p. 2]. This literature survey will inform the development of an AR/TUI-based tool in order to support two planes of collaborative programming: students’ interaction with program visualisation and social and communicative processes that need to be included.

At present, plans for the development of a collaborative augmented reality environment prototype are underway to observe the effect that AR and TUIs may have on tertiary student collaborative learning spaces within programming courses, and on AR-supported visualisation of programming concepts and code debugging. Such an environment will comprise an AR-supported TUI with which student programming pairs may freely interact, combined with the traditional two-dimensional on-screen code representation (Jeliot GUI). This may provide an additional layer of interactivity by dynamically updating based on physical proximity and spatial positioning of physical manipulatives representing code snippets or individual elements of pseudo-code. Students are thus invited to engage with the program code presented within Jeliot through a primarily tangible medium, allowing them to understand the way in which changes can present on both a visual model and a syntax-based one.

The design of a workspace for collaborative code debugging will comprise three key elements: interactive source code with program visualisation represented by the Jeliot 3 environment (GUI); tangible manipulatives for interfacing with augmented code visualisation; and textual annotation viewable as AR content. Physical markers, which form the core of the TUI, present with simple paper manipulatives arranged in a linear fashion. Each marker is appended with AR-based visual cues comprising textual information, iconography, and animated visualisation, which serves an explanatory role for programming pairs working with Jeliot code. By studying the visual information provided and interacting with it, users can
obtain indirect guidance in debugging code errors and understanding overall code structure.

REFERENCES


